Java

Senthil Natarajan

Please Join in below link

Instagram: https://www.instagram.com/rba_infotech/
LinkedIn: https://www.linkedin.com/company/rba-infotech/
Facebook:
https://www.facebook.com/people/RBA-Infotech/100043552406579/

Introduction to Java

- **It is a Object Oriented Programming (OOPs) language which was invented by James Gosling in 1995 in Sun Microsystem, later it became subsidy of Oracle. Before Java it was named Oak.**

- **Advantages of Java**
  - **Object Oriented** − In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
  - **Platform Independent** − Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
  - **Simple** − Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
  - **Secure** − With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
  - **Architecture-neutral** − Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
  - **Portable** − Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
  - **Robust** − Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.

Procedural vs Object Oriented Programming Language

PROCEDURAL ORIENTED PROGRAMMING	**OBJECT ORIENTED PROGRAMMING**
In procedural programming, program is divided into small parts called ***functions***.	In object oriented programming, program is divided into small parts called ***objects***.
Procedural programming follows ***top down approach***.	Object oriented programming follows ***bottom up approach***.
There is no access specifier in procedural programming.	Object oriented programming have access specifiers like private, public, protected etc.
Adding new data and function is not easy.	Adding new data and function is easy.

Contd…

PROCEDURAL ORIENTED PROGRAMMING	**OBJECT ORIENTED PROGRAMMING**
Procedural programming does not have any proper way for hiding data so it is ***less secure***.	Object oriented programming provides data hiding so it is ***more secure***.
In procedural programming, overloading is not possible.	Overloading is possible in object oriented programming.
In procedural programming, function is more important than data.	In object oriented programming, data is more important than function.
Procedural programming is based on ***unreal world***.	Object oriented programming is based on ***real world***.
Examples: C, FORTRAN, Pascal, Basic	Examples: C++, Java, Python, C# etc.

C++ vs Java

COMPARISON PARAMETER	C++	JAVA
Developed / Founded by	C++ was developed by Bjarne Stroustrup at Bell Labs in 1979. It was developed as an extension of the C language.	Java was developed by James Gosling at Sun Microsystems. Now, it is owned by Oracle.
Programming model	It has support for both procedural programming and object-oriented programming.	Java has support only for object-oriented programming models.
Platform dependence	C++ is platform dependent. It is based on the concept of Write Once Compile Anywhere.	Java is platform-independent. It is based on the concept of Write Once Run Anywhere.

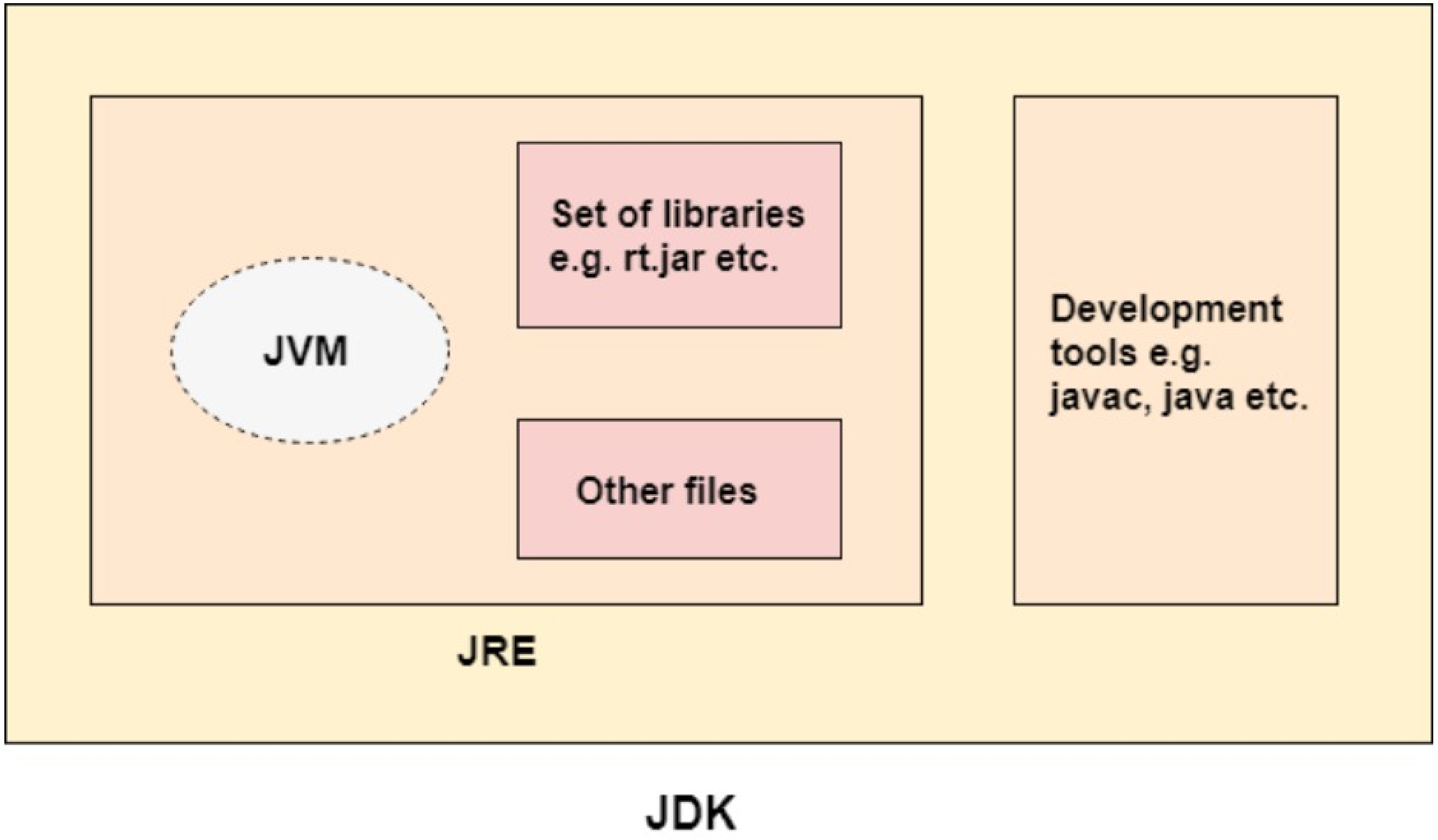
C++ vs Java

COMPARISON PARAMETER	C++	JAVA
Features supported	C++ supports features like operator overloading, Goto statements, structures, pointers, unions, etc.	Java does not support features like operator overloading, Goto statements, structures, pointers, unions, etc.
Compilation and Interpretation	C ++ is only compiled and cannot be interpreted.	Java can be both compiled and interpreted.
Library and Code reusability support	C++ has very limited libraries with low-level functionalities. C++ allows direct calls to native system	Java, on the other hand, has more diverse libraries with a lot of support for code reusability. In Java, only calls through the Java Native Interface and recently Java

C++ vs Java

COMPARISON PARAMETER	C++	JAVA
Memory Management	In C++, memory management is manual.	In Java, memory management is System controlled.
Type semantics	C++ is pretty consistent between primitive and object types.	In Java, semantics differs for primitive and object types.
Global Scope	In C++, both global and namespace scopes are supported.	Java has no support for global scope.
Access control and object protection	In C++, a flexible model with constant protection is available.	In Java, the model is cumbersome and encourages weak encapsulation.

JDK vs JRE vs JVM


Set of libraries e.g. rt.jar etc.
JVM
Other files
Development tools e.g. javac, java etc.
JRE
JDK

JVM Architecture

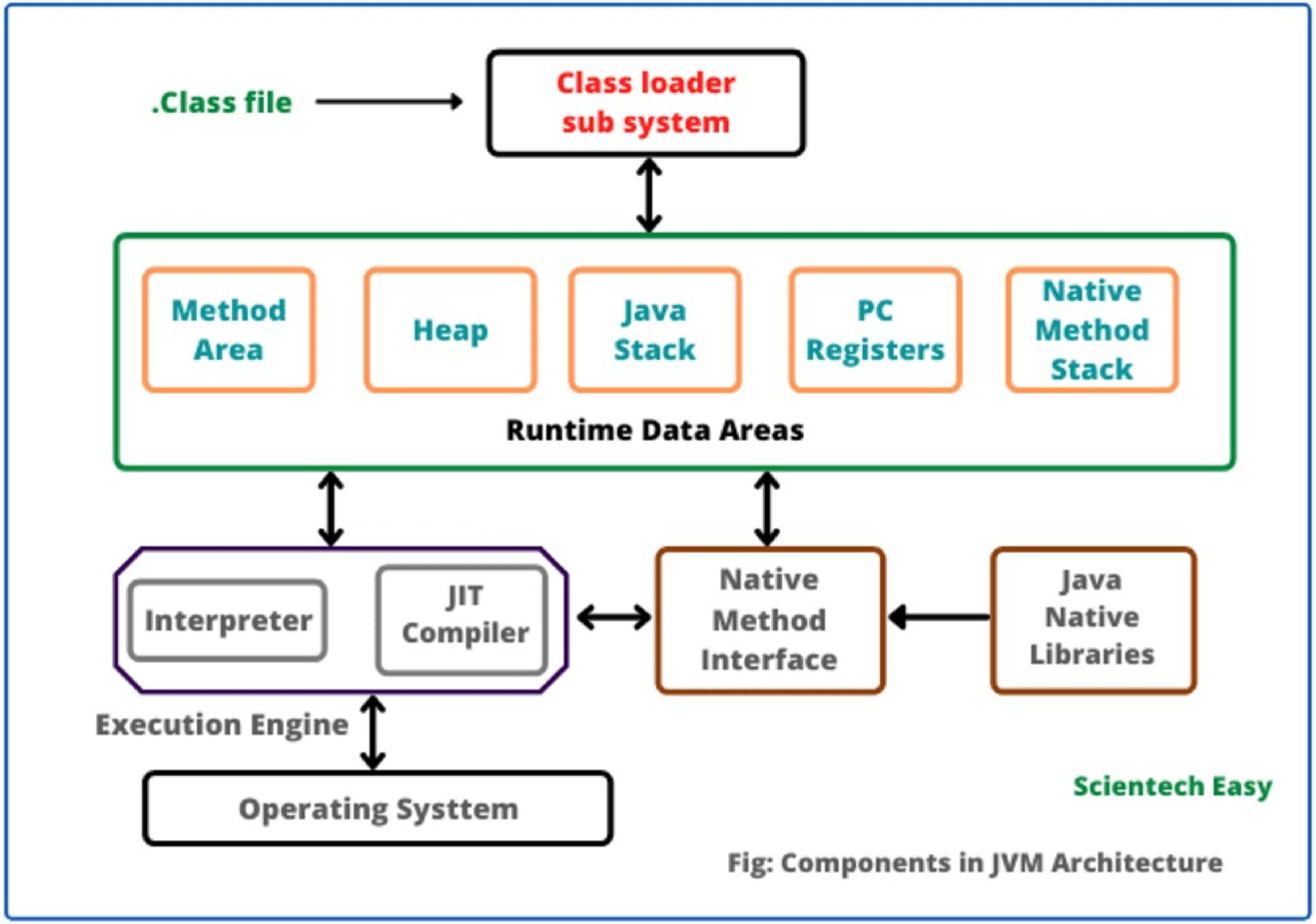


Fig: Components in JVM Architecture

Structure of Java Program

- Documentation Section
- Package Statement
- Import Statements
- Class Definition
- Main Method Class
  - Main Method Definition

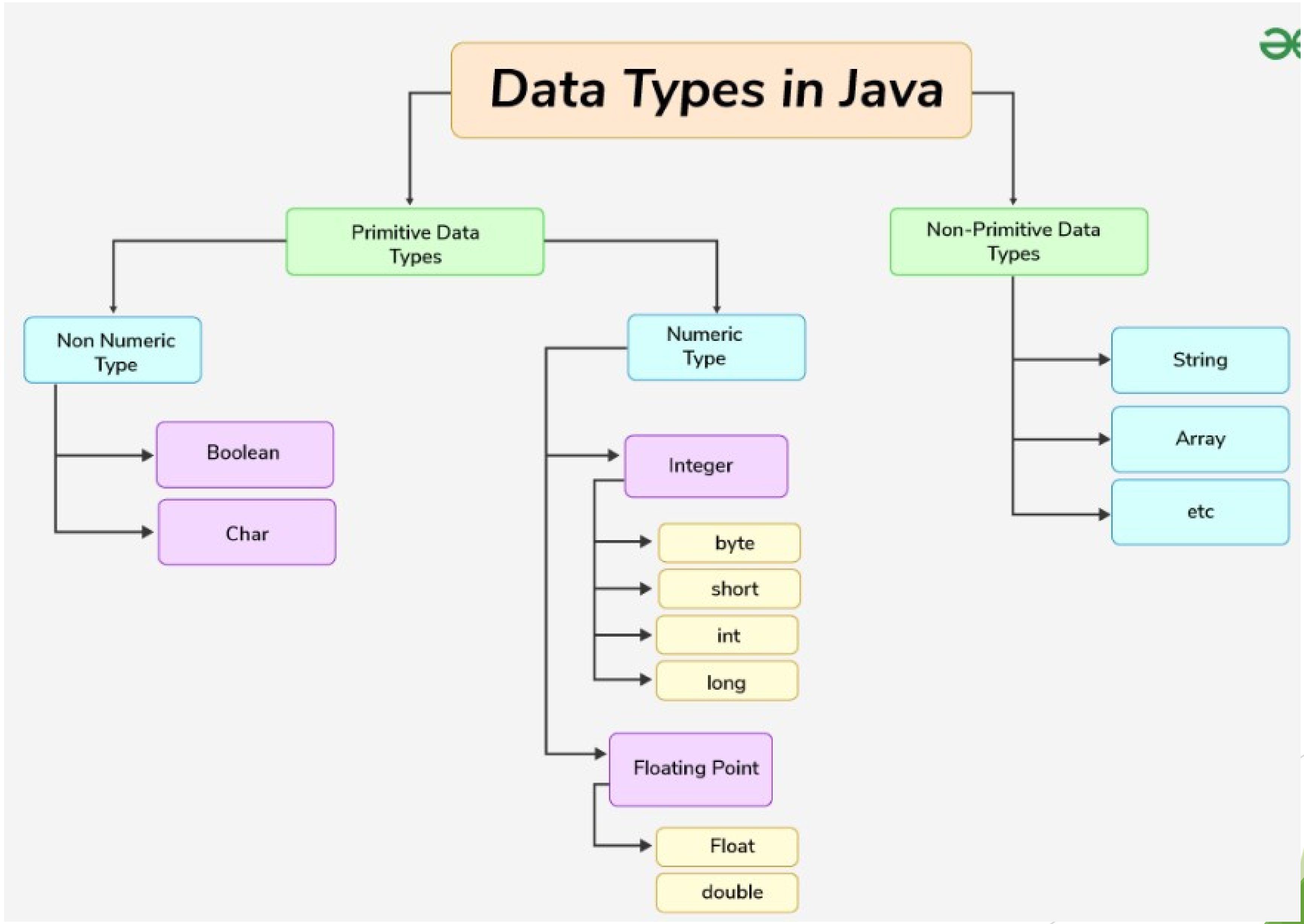
Structure of Java Program

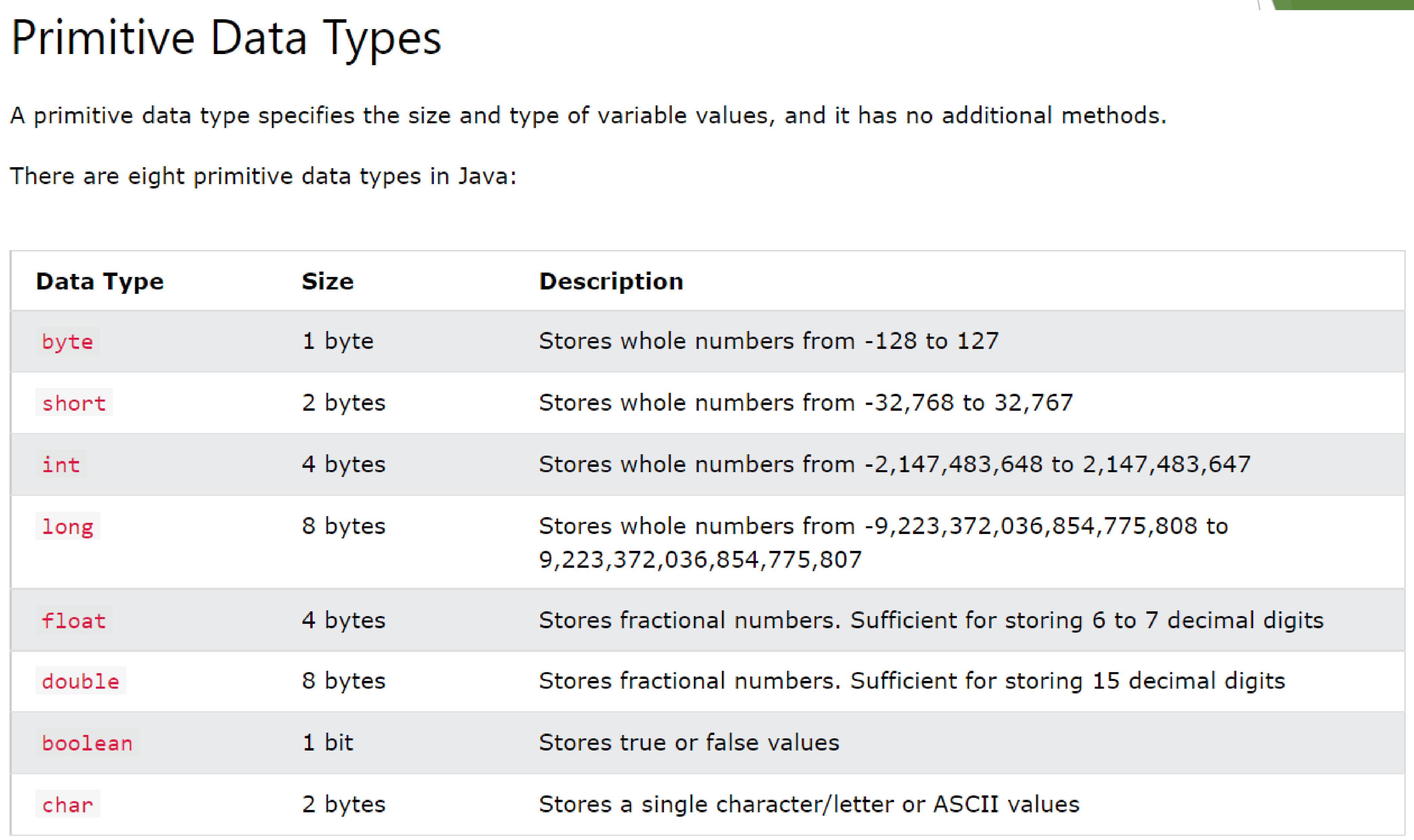
Ex.

Hello.java

```
        /* Author: Senthil
        Date: 202-01-07
        Description: Writes the words "Hello Java" on the screen */
public class Hello
{
        public static void main(String[] args)
        {
         System.out.println("Hello Java");
        }
 }
```

Data Types


Data Types in Java
Primitive Data Types
Non-Primitive Data Types
Non Numeric Type
Numeric Type
Boolean
Char
Integer
byte
short
int
long
Floating Point
Float
double
String
Array
etc



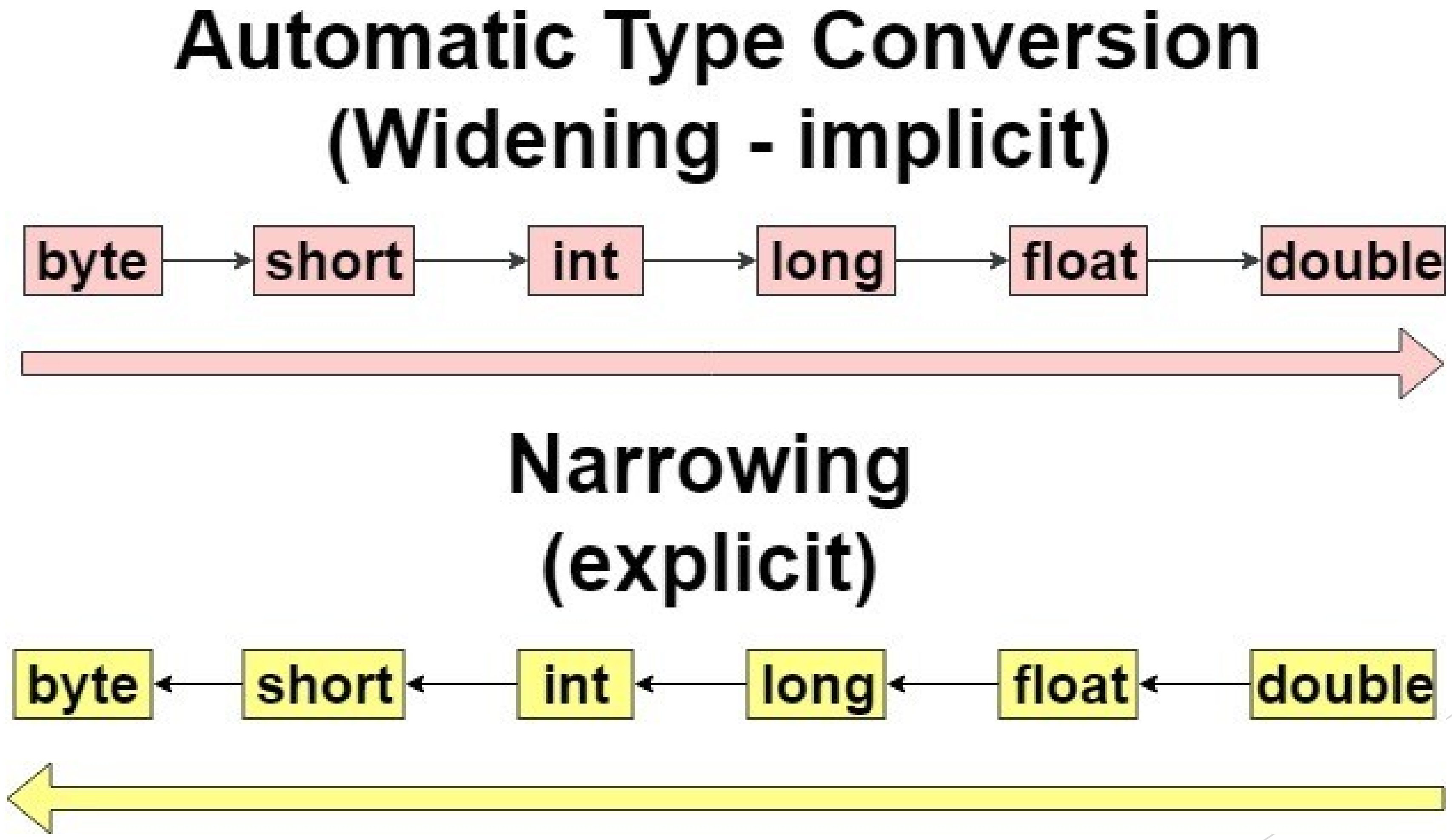
Primitive Data Types

A primitive data type specifies the size and type of variable values, and it has no additional methods.

There are eight primitive data types in Java:

Data Type	Size	Description
`byte`	1 byte	Stores whole numbers from -128 to 127
`short`	2 bytes	Stores whole numbers from -32,768 to 32,767
`int`	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
`long`	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`float`	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
`double`	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
`boolean`	1 bit	Stores true or false values
`char`	2 bytes	Stores a single character/letter or ASCII values

Type Casting



Java Naming Conventions

Class

It should start with the uppercase letter. In case of more than one words first letter of each word should be capital letter.

It should be a noun such as Color, Button, System, Thread, etc.

Use appropriate words, instead of acronyms.

Example: -

```
public class EmployeeDetails
{
//code snippet
}
```

Java Naming Conventions

Variable

It should start with a lowercase letter such as id, name.

It can start with only two special characters $ (dollar) and _ (underscore), other special characters are not allowed.

If the name contains multiple words, **first letter of first word should be small case** and first letter of remaining words should be **upper case** such as firstName, lastName.

Avoid using one-character variables such as x, y, z.

Example :-

```
 class Employee
{
//variable
int id;
Int empNo;
//code snippet
}
```

Java Naming Conventions

Method

It should start with lowercase letter.

It should be a verb form such as main(), print(), println().

If method name is multiple words, first letter of the first word should be in small case and first letter of the remaining words should be upper case.

e.x: actionPerformed().

Example:-

```
 class Employee
{
//method
void draw()
{
//code snippet
}
}
```

Java Naming Conventions

Constant

It should be in uppercase letters such as RED, YELLOW.

If the name contains multiple words, it should be separated by an underscore(_) such as MAX_PRIORITY.

It may contain digits but not as the first letter.

Example :-

```
class Employee
{
//constant
 static final int MIN_AGE = 18;
//code snippet
}
```

Java Naming Conventions

Interface

It should start with the uppercase letter.

It should be an adjective such as Runnable, Remote, ActionListener.

Use appropriate words, instead of acronyms.

Example: -

```
interface Printable
{
//code snippet
}
```

Java Naming Conventions

Package

It should be a lowercase letter such as java, lang.

If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang, com.demo

Example :-

package com.javatpoint; //package

class Employee

{

//code snippet

}

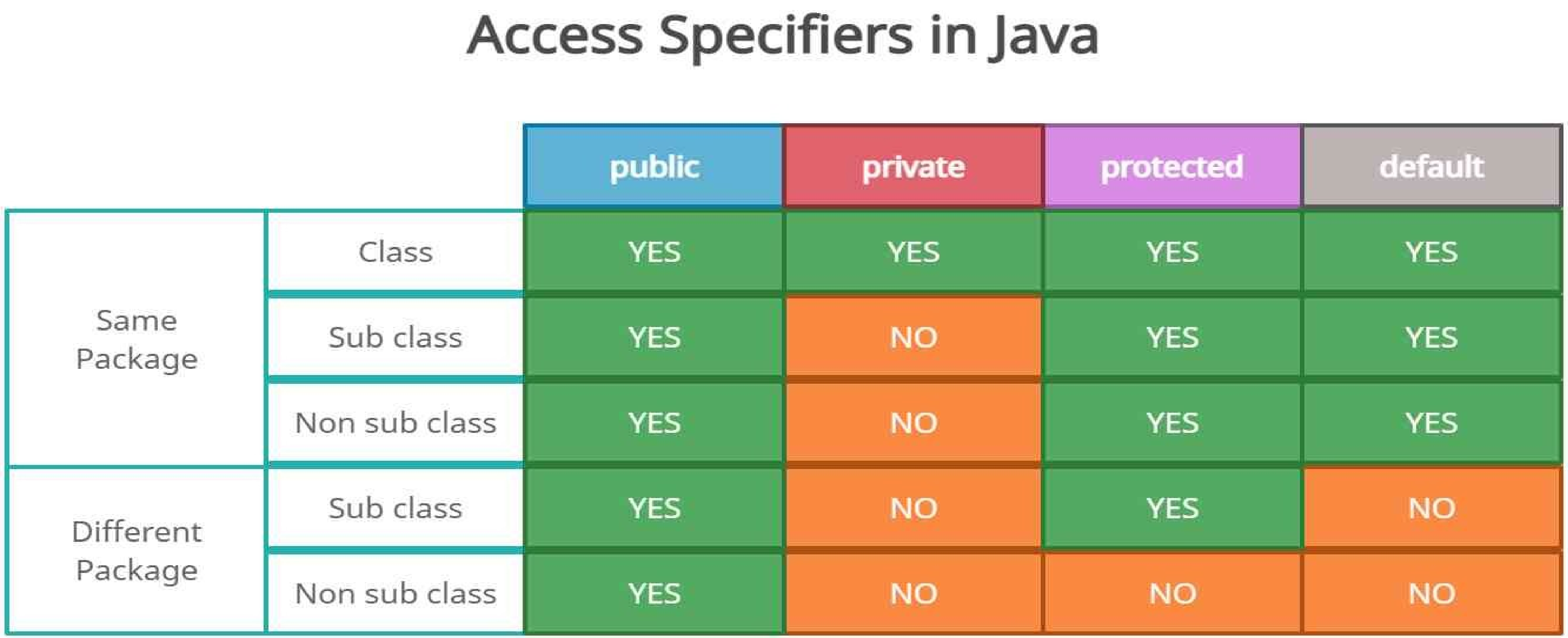
Java Naming Conventions

CamelCase in java naming conventions

Java follows camel-case syntax for naming the class, interface, method, and variable.

If the name is combined with two words, the second word will start with uppercase letter always such as actionPerformed(), firstName, ActionEvent, ActionListener, etc.

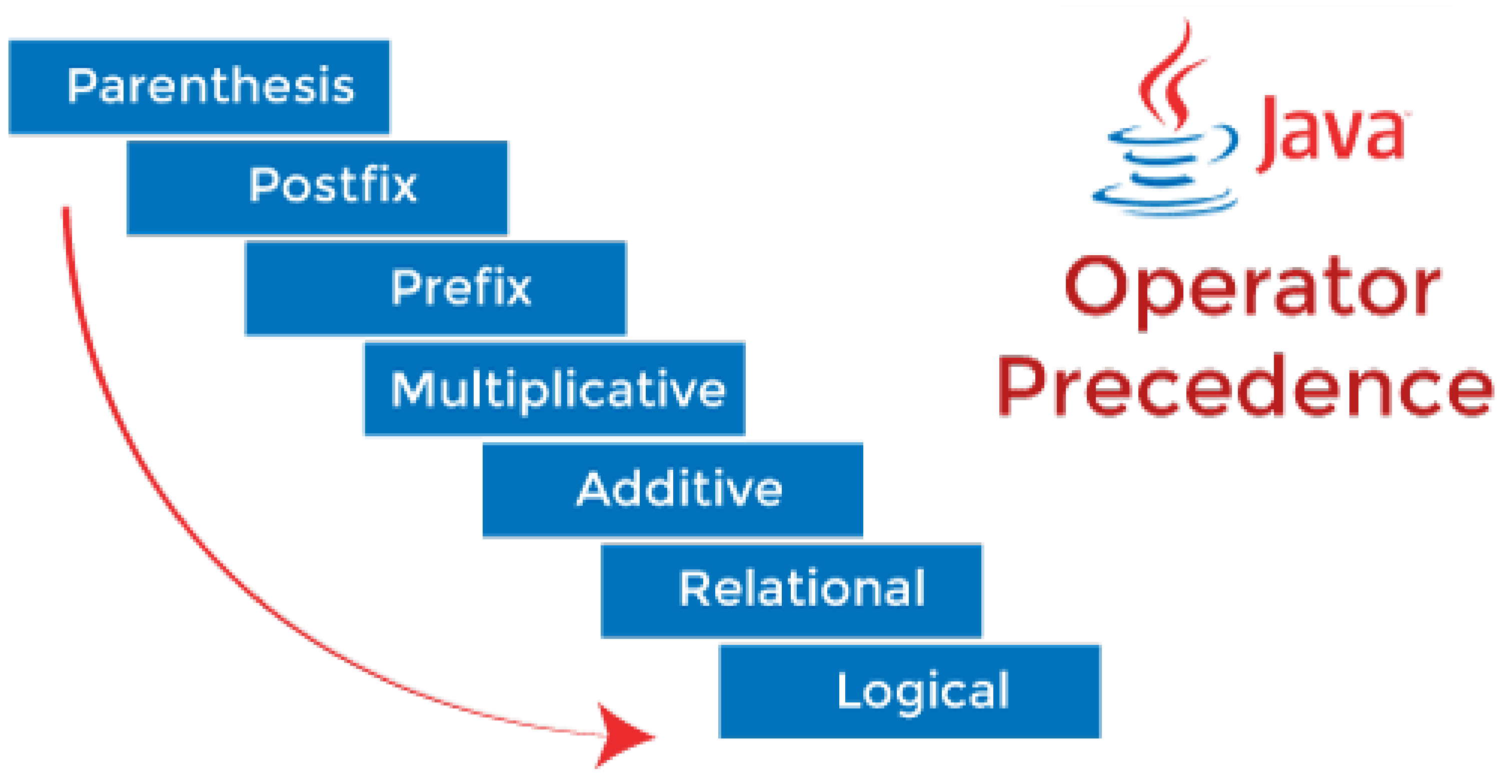
Access Modifiers / Access Specifiers



Access Specifiers in Java

		public	private	protected	default
Same Package	Class	YES	YES	YES	YES
	Sub class	YES	NO	YES	YES
	Non sub class	YES	NO	YES	YES
Different Package	Sub class	YES	NO	YES	NO
	Non sub class	YES	NO	NO	NO

Operators Precedence



Operators

Operator Type	Category	Precedence
1. Arithmetic	Multiplicative	*, /, %
	Addictive	+, -
2. Relational	Comparison	<, <=, >, >=
	Equality	==, !=
3. Logical	And	&&
	Or	||
	Not	!
4. Unary	Postfix	expr++, expr– Ex. i++, j--
	Prefix	++expr, --expr Ex. --i, --j
5. Ternary		(condition)? Stt : Stt
6. Bitwise	Bitwise And	&
	Bitwise Or	|
	Bitwise Exclusive Or	^
7. Shift		<<, >>
8. Assignment		=, +=,-+,*=,/+,%=, &=, !=, ^=, <<=, >>=

Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (type)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right

Precedence	Operator	Type	Associativity
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right

Precedence	Operator	Type	Associativity
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division	Right to left

Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (type)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left

Statements

Statements

1. If else
2. For loop
3. For each
4. While loop
5. Do … while loop
6. Switch… case
7. Break
8. Continue

For each loop

1. is another array traversing technique like for loop, while loop, do-while loop
2. Syntax:

```
for( datatype var : array or collection)
{
Stt;
}
```

Java String

String

- String is collection of characters.

  Ex:

  ```
  String s1 = “Selenium”
  String s2 = “Selenium”
  ```

The String is immutable. The String value cannot be changed.

How to create a string object?

There are two ways to create String object.

1. By string literal

   Ex:

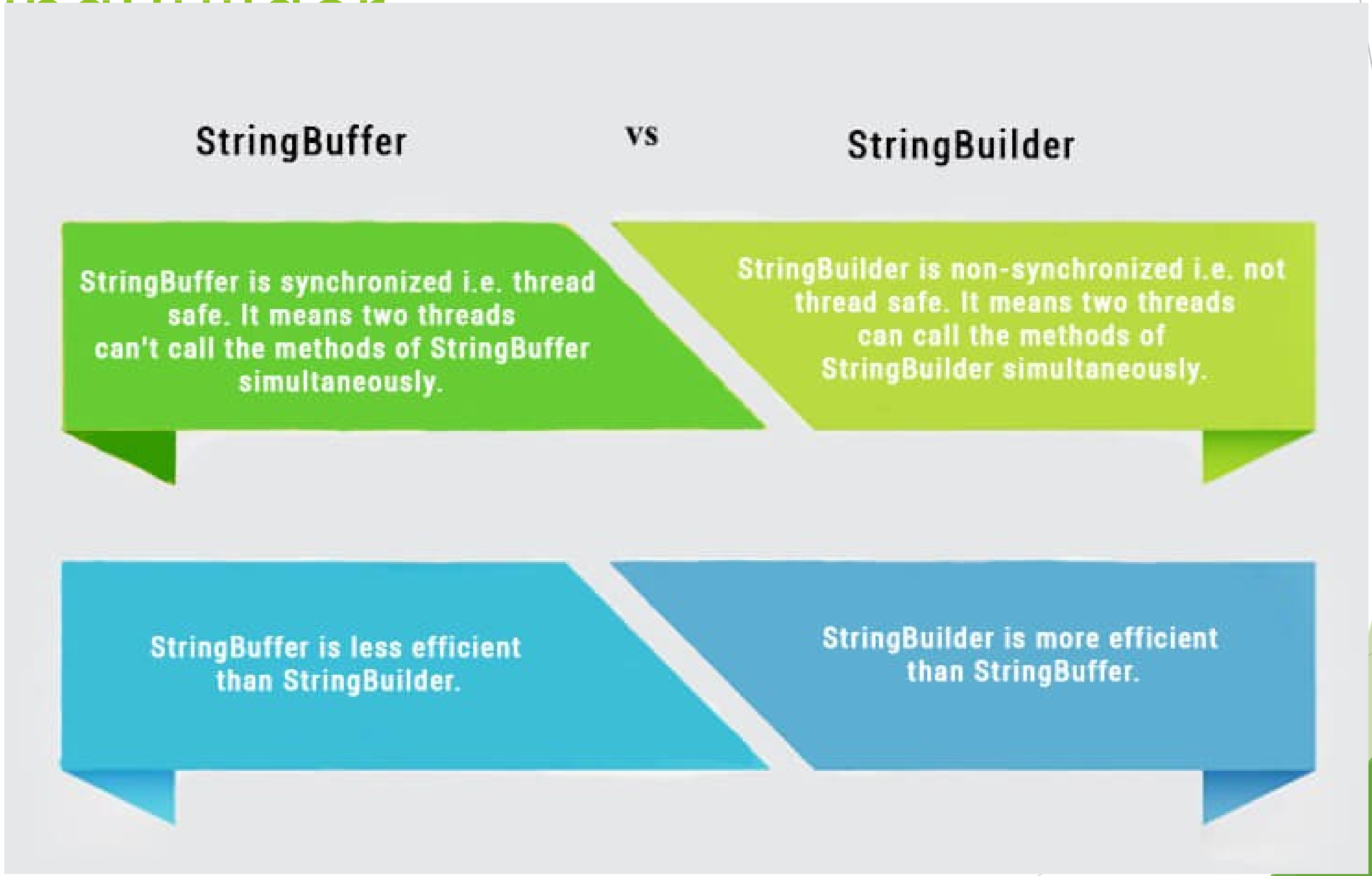
   ```
   String s1 = “Selenium”
   ```

2. By new keyword

   Ex:

   ```
   String s2 = new String(“Selenium”);
   ```

String vs StringBuffer vs Str



StringBuffer vs StringBuilder

StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.

StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.

StringBuffer is less efficient than StringBuilder.

StringBuilder is more efficient than StringBuffer.

Compare Two Strings

1. By equals() method

   Compares content of two strings

2. By = = operator

   Compares reference not values

3. By compareTo() method

   Compares values lexicographically and returns an integer value

Arrays

- Collection of **Similar data types**
- There are two types of array.
    1. Single Dimensional Array
    2. Multidimensional Array

Syntax to Declare Array:

```
dataType[] arr; (or)
dataType []arr; (or)
dataType arr[];
```

Syntax to Instantiation of an Array in Java

```
arrayRefVar=new datatype[size];
```

Syntax to Declare and Instantiate of an Array

```
datatype[] arrayRefVar=new datatype[size];
```

OOPs – Object Oriented Programming

The following 4 concepts are the features of the OOPs

1. **Class**
2. **Object**
3. **Encapsulation**
4. **Inheritance**
5. **Polymorphism**
6. **Abstraction**
    1. **Abstract Class**
    2. **Interface**

OOPs – Object Oriented Programming

The few other concepts of the OOPs

1. **Constructor**
2. **Package**
3. **Static**
4. **Final**
5. **this keyword**
6. **super**

OOPs – Object Oriented Programming

The following 4 concepts are the features of the OOPs

1. **Abstraction**
2. **Encapsulation**

    Binding the data member and member functionality into single unit is called

3. **Inheritance**

    Deriving the properties from one class to anther class

4. **Polymorphism**

    Performing many actions with one name

    1. Method Overloading

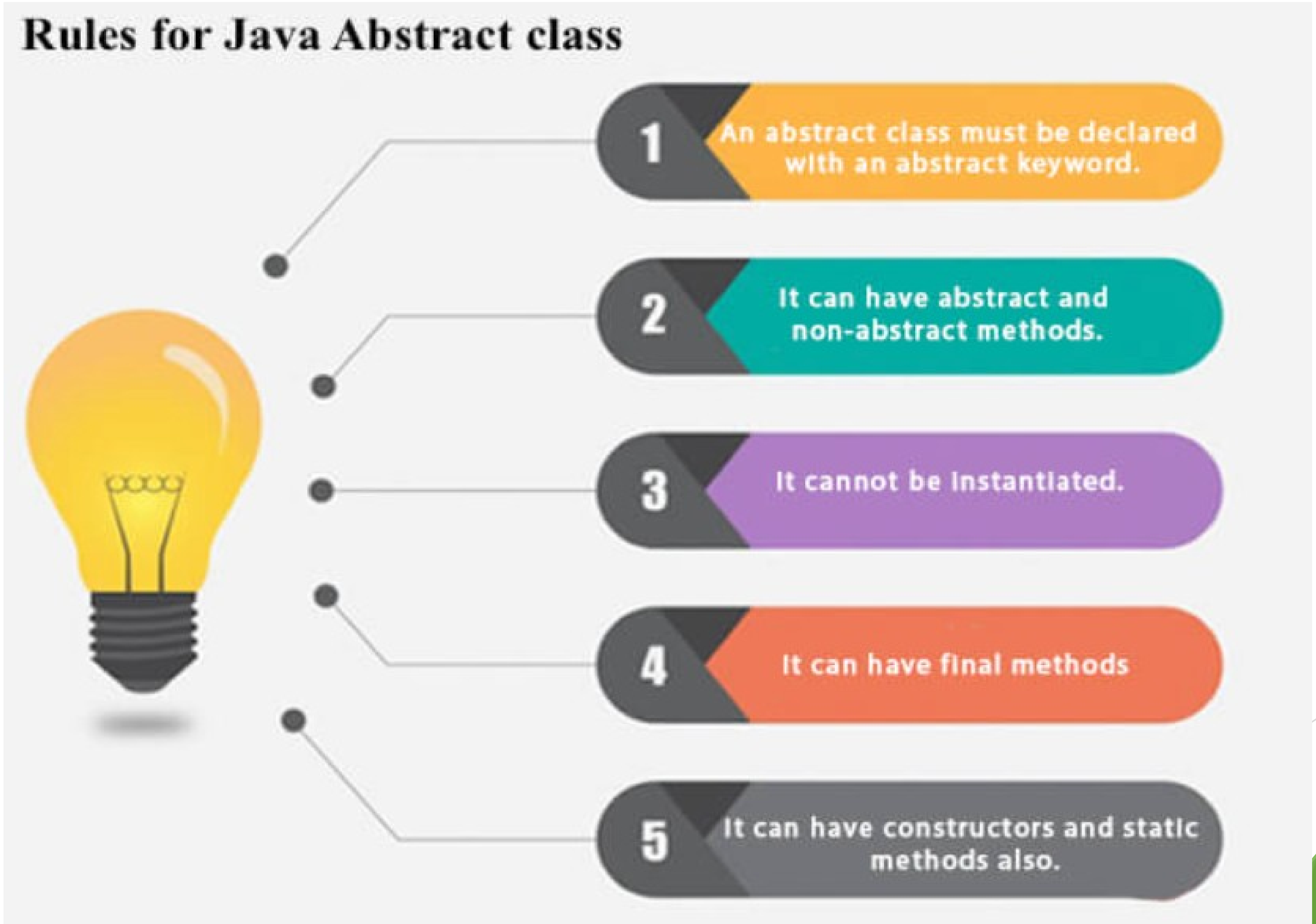
    2. Method Overriding

Abstract Class and Interface

Data Abstraction

1. **Abstract Class – Partial Abstraction (0 -100%)**
2. **Interface – Full Abstraction (100%)**

Abstract Class



Rules for Java Abstract class

1. An abstract class must be declared with an abstract keyword.
2. It can have abstract and non-abstract methods.
3. It cannot be instantiated.
4. It can have final methods
5. It can have constructors and static methods also.

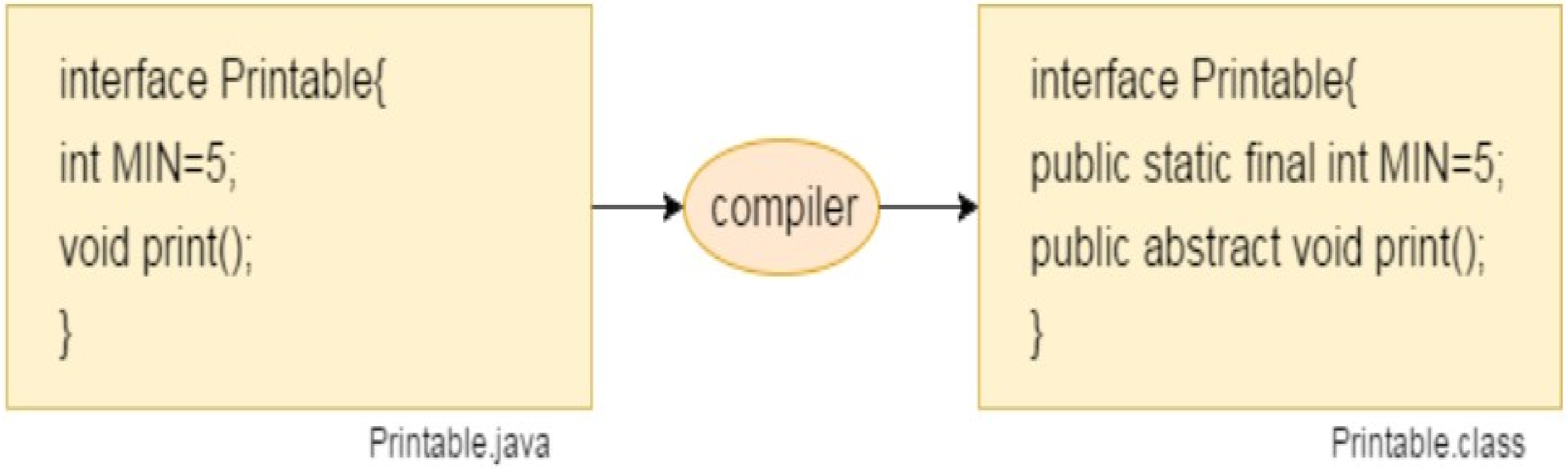
Interface

What is Interface?

It is contains variable and only abstract methods.

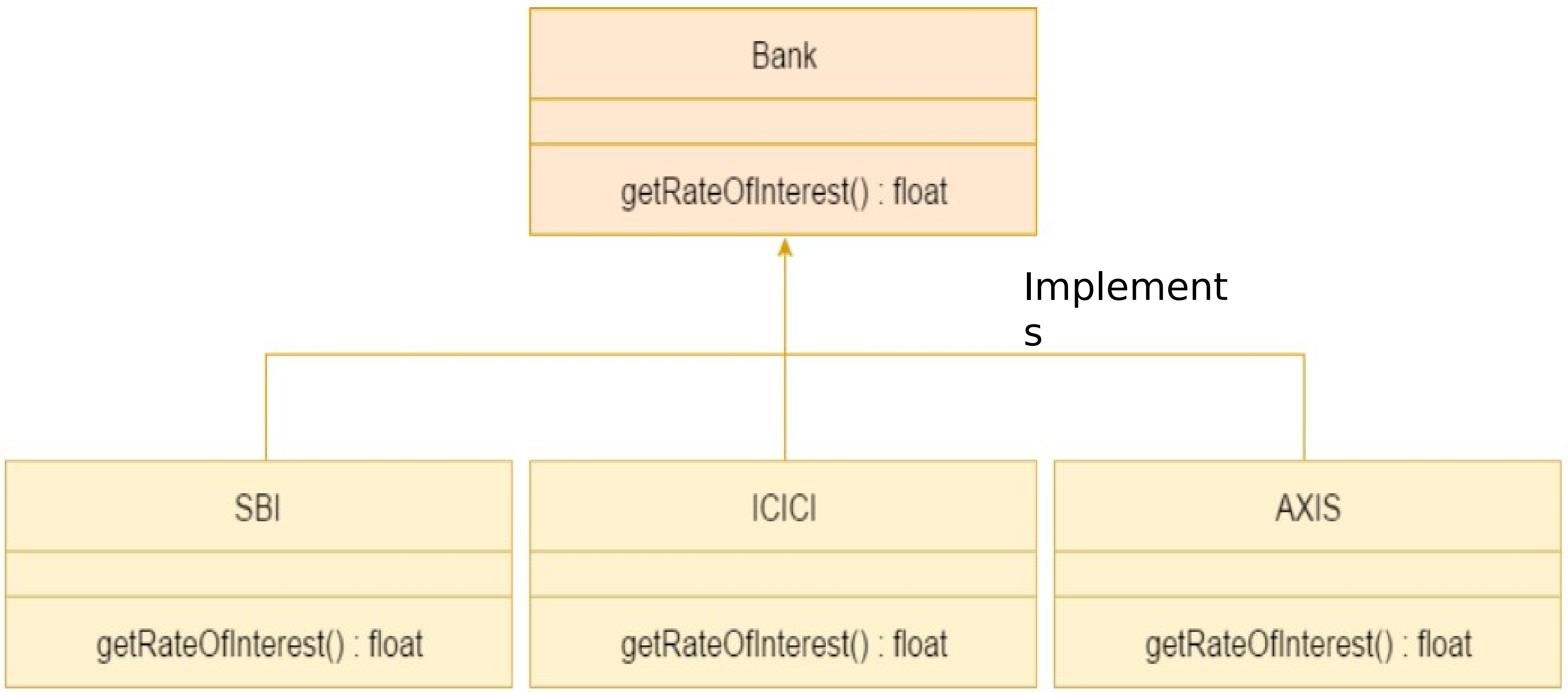
```
public interface Employeeable {
String empName="Senthil";
void showEmpDetails();

}
```

Interface



```
interface Printable{
int MIN=5;
void print();
}
```

```
interface Printable{
public static final int MIN=5;
public abstract void print();
}
```


Bank
getRateOfInterest() : float
SBI
getRateOfInterest() : float
ICICI
getRateOfInterest() : float
AXIS
getRateOfInterest() : float

Implement
s

Abstract Class vs Interface

Abstract class	**Interface**
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private,	Members of a Java interface are public by default.

Package

What is Package?

Package is the pack of classes, interfaces and other packages.

Two Types of Package:

1. **Pre-defined or Built-in Packages**
2. **User-Defined Packages**

<u>Build-In Package:</u>

java.util.Scanner;

Here:
→ **java** is a top level package
→ **util** is a sub package
→ and **Scanner** is a class which is present in the sub package **util**.

Package

Why Packages?

1. **Reusability**
2. **Better Organization**
3. **Name Confilicts**

Ways to access the Package from another Package

4. import package.*;

   Ex: import java.util.*;

5. import package.classname;\

   import java.util.Scanner;

6. fully qualified name.

   java.util.Scanner sc = new java.util.Scanner();

Exception Handling

What is Exception?

- An Exception is an **unexpected event that interrupts the normal execution flow of the program**. When an exception occurs program execution gets terminated.
- In such cases we get a system generated error message.
- The good thing about exceptions is that **they can be handled** in Java.
- By handling the exceptions we can provide a meaningful message to the user about the issue rather than a system generated message, which may not be understandable by user.

Exception Handling

Why Exception occurs?

- There can be several reasons that can cause a program to throw exception.
- For example:
    1. Opening a non-existing file in your program
    2. Bad input data provided by user.
    3. Accessing the array elements out of the index range.

Syntax for Handling Exception

Syntax:

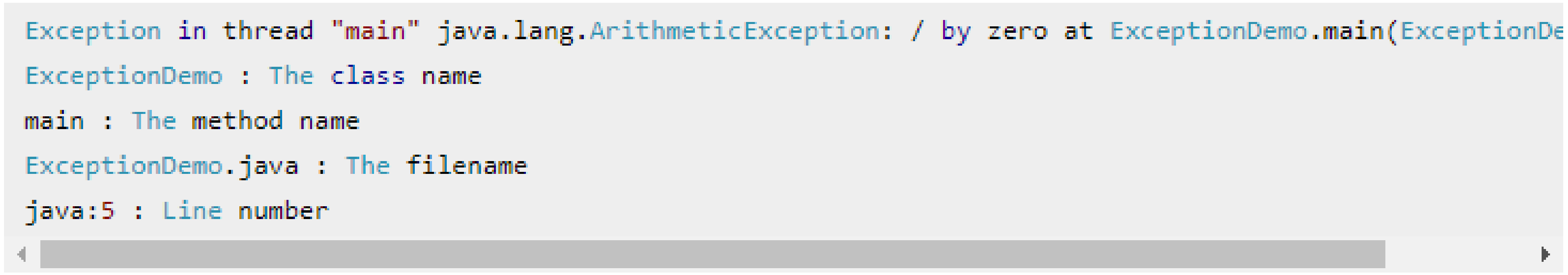
```
try{
// code snippet which causes exception
}catch(approriate exception a){
System.out.println("Exception occured");
}
```

Ex:

```
    try{
    A = 10/0;
    }catch(ArithmeticException ae){
    System.out.println("Divided by zero exception occured");
    }
```

System vs User generated Exception Message

The below is the system generated exception



```
Exception in thread "main" java.lang.ArithmeticException: / by zero at ExceptionDemo.main(ExceptionDe
ExceptionDemo : The class name
main : The method name
ExceptionDemo.java : The filename
java:5 : Line number
```

The below is the user defined exception message

The number is divided by zero on line no 5.

Types of Exception

Two Types:

1. Checked Exception / Compile time exception
2. Unchecked Exception / Runtime exception

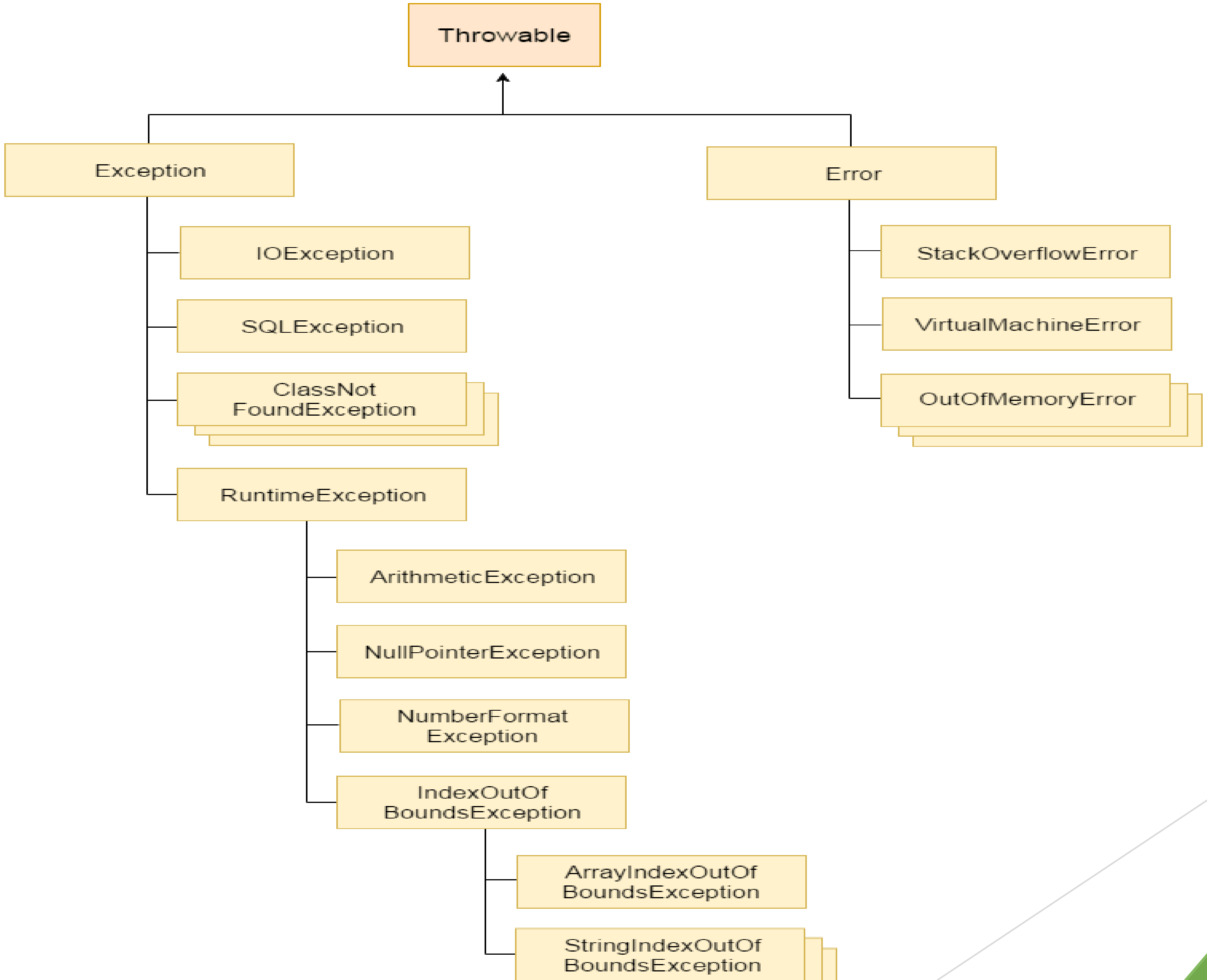
Checked Exception

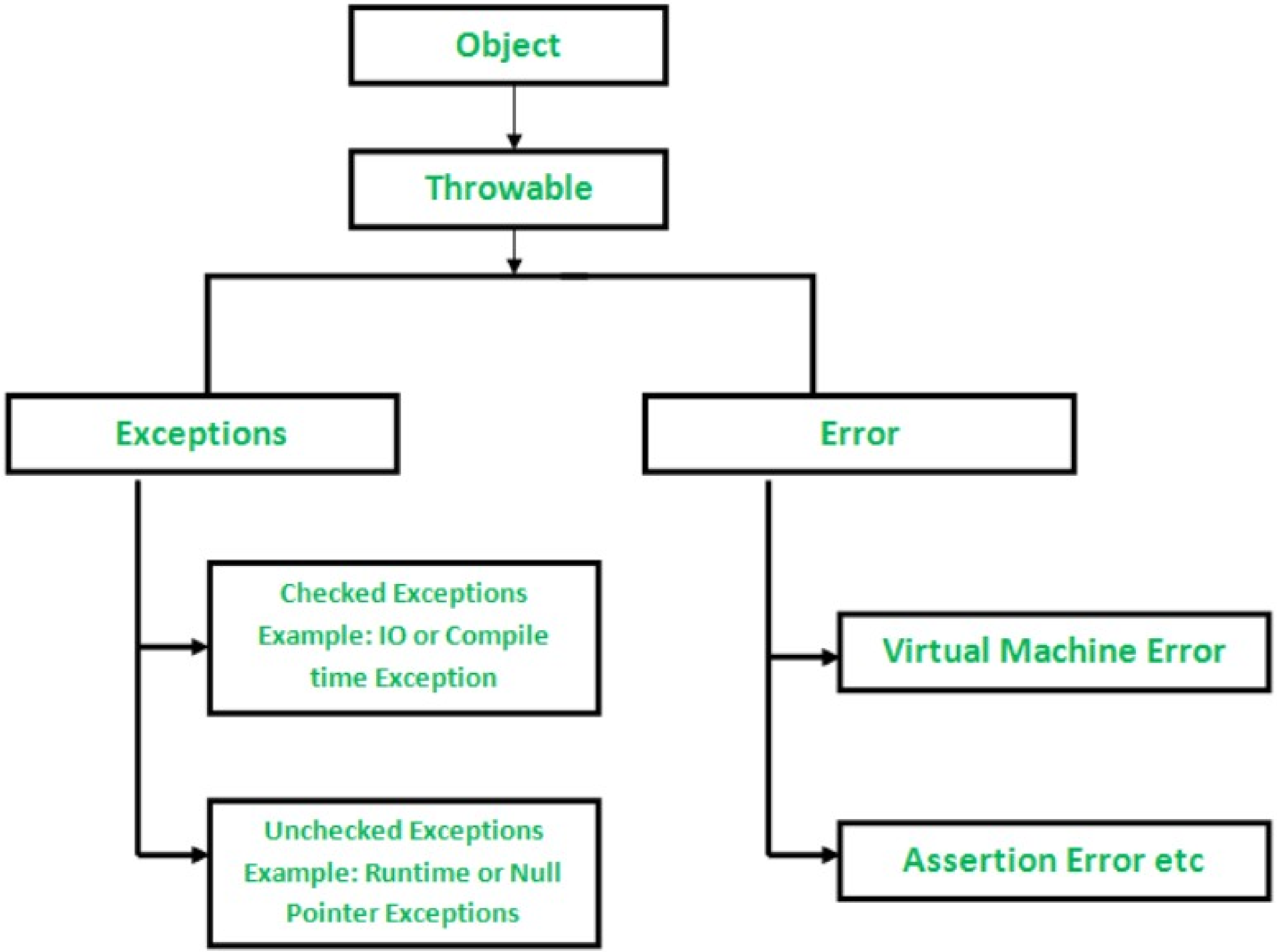
- The compiler checks the programme during compilation to see whether the programmer has handled them or not.
- If these exceptions are not handled/declared in the program, you will get compilation error. For example, SQLException, IOException, ClassNotFoundException etc
- All exceptions other than Runtime Exceptions are known as Checked exceptions as

Unchecked Exception

- These exceptions are not checked at compile-time so compiler does not check whether the programmer has handled them or not.
- Runtime Exceptions are also known as Unchecked Exceptions.

Hierarchy of Exception Class




Object
Throwable
Exceptions
Error
Checked Exceptions Example: IO or Compile time Exception
Unchecked Exceptions Example: Runtime or Null Pointer Exceptions
Virtual Machine Error
Assertion Error etc

Difference between Error and Exception

Error:

- Indicate that something severe enough has gone wrong.
- The application should crash rather than try to handle the error.
  - Ex:
  - Memory out of the error
  - Stack Trace Error

Exceptions

- They are events that occurs in the code.
- A programmer can handle such conditions and take necessary corrective actions
- E.x:
  1. Arithmetic Exception
  2. IOException
  3. NullPointer Exception

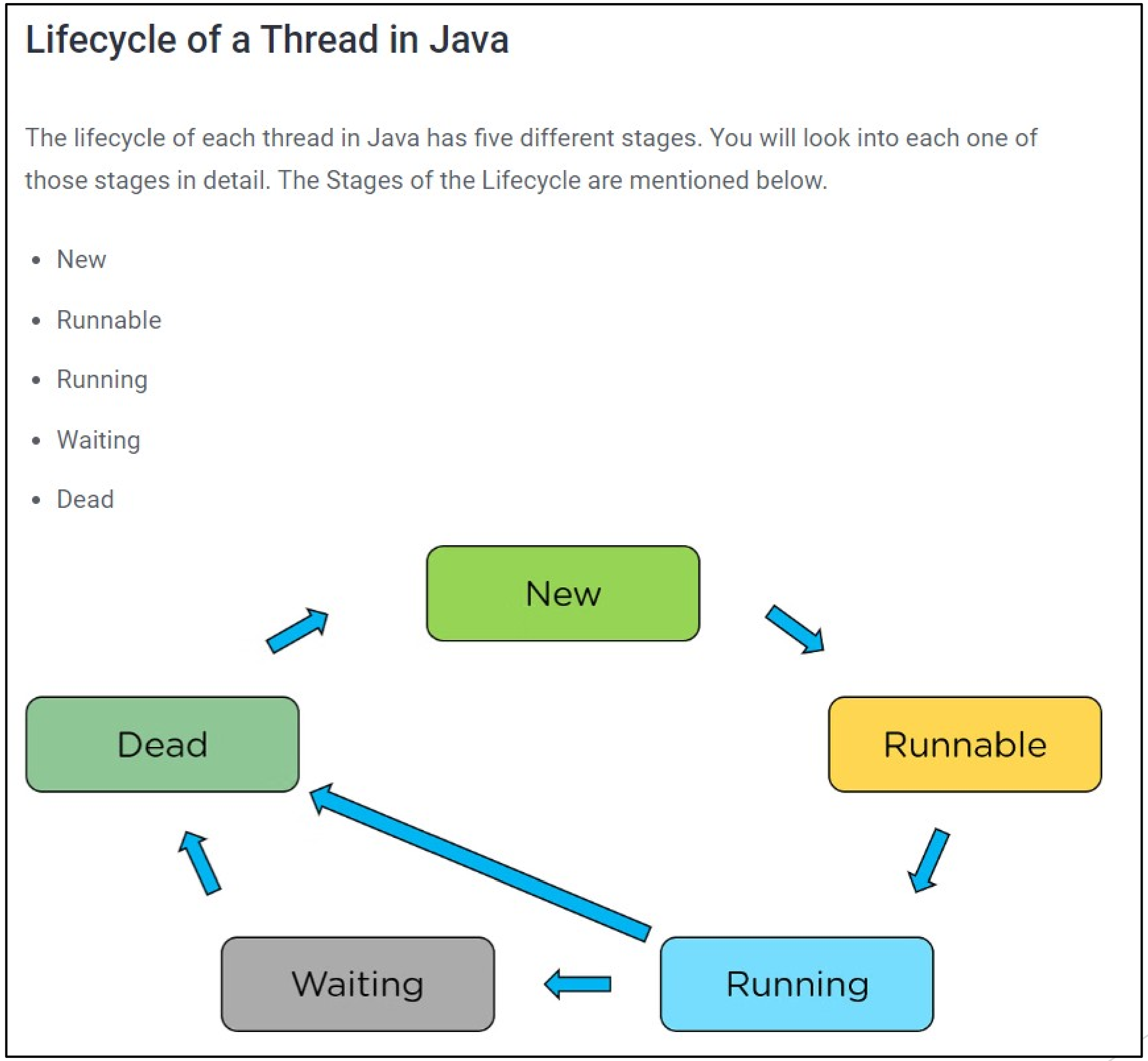
Multithreading

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

hreads can be created by using **two mechanisms** :

1. Extending the Thread class
2. Implementing the Runnable Interface

Thread Life Cycle



Lifecycle of a Thread in Java

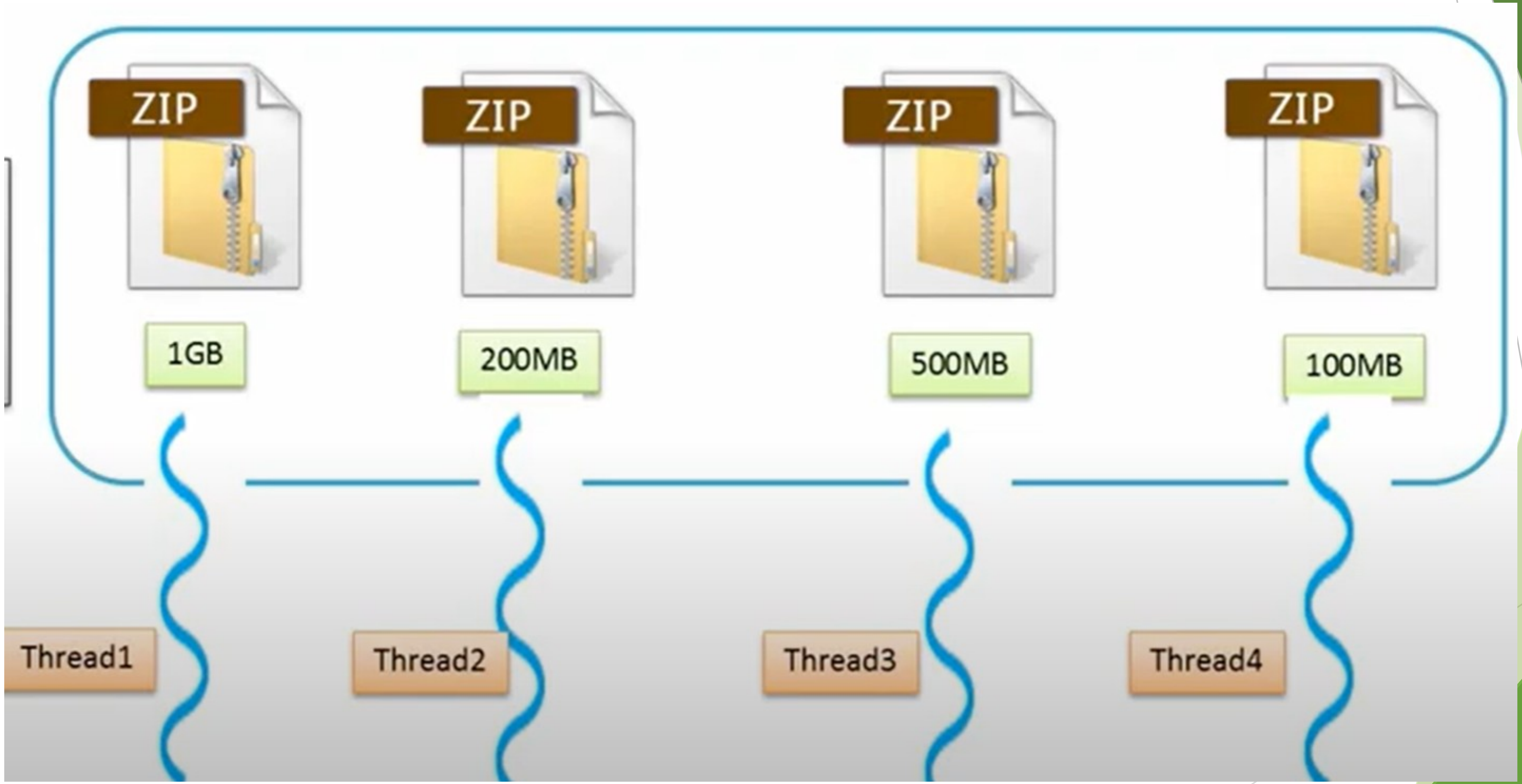
The lifecycle of each thread in Java has five different stages. You will look into each one of those stages in detail. The Stages of the Lifecycle are mentioned below.

- New
- Runnable
- Running
- Waiting
- Dead

$$\begin{tikzcd}
 & New \arrow[dr] & \\
Dead \arrow[ur] & & Runnable \arrow[d] \\
Waiting \arrow[u] & & Running \arrow[ll] \arrow[ull]
\end{tikzcd}$$

Real Time Example

Another Example



Collections

What is Collections?

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects. Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Why Collections?

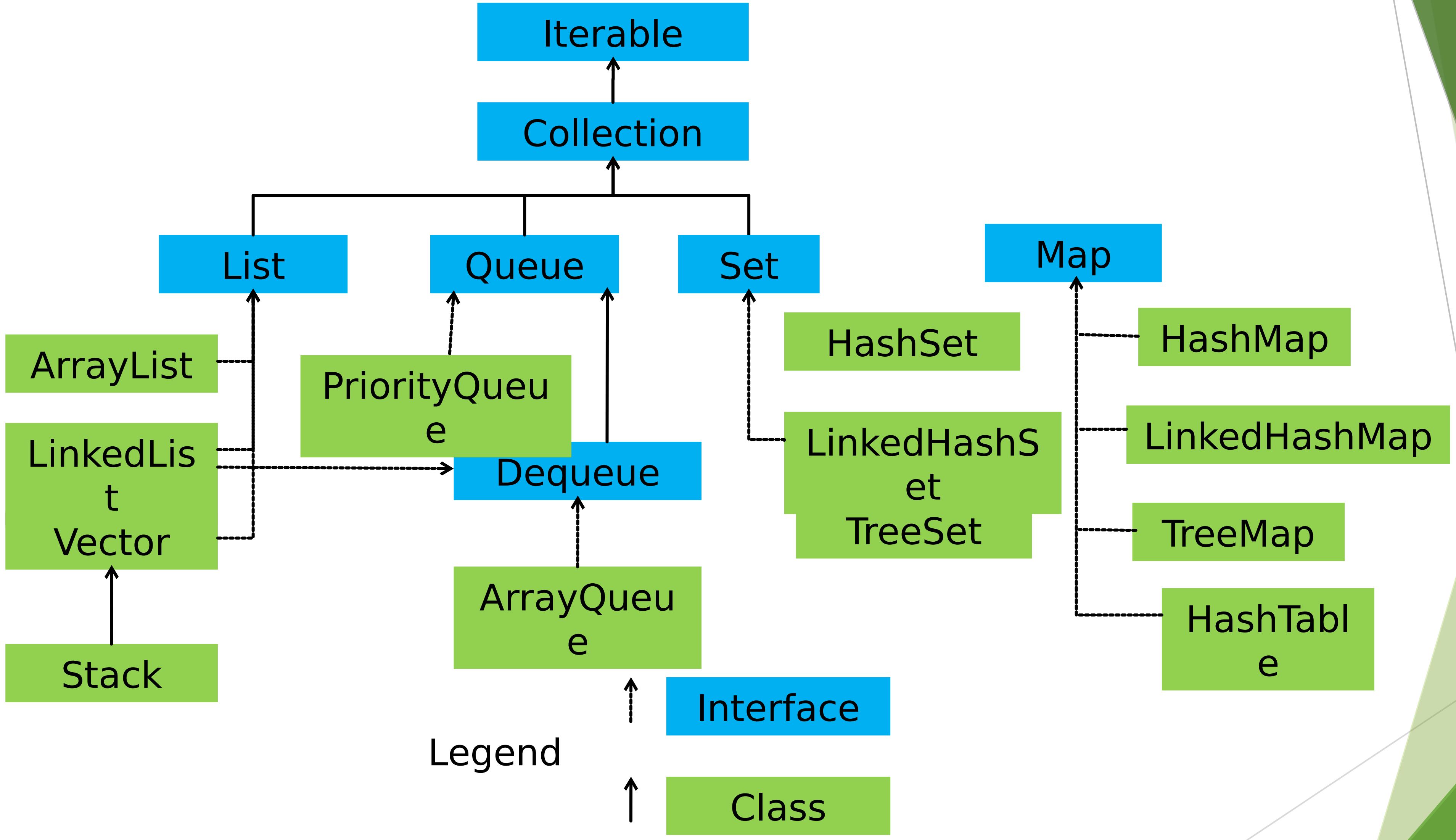
1. To handle group of objects
2. It provides dynamically growing or shrinking size whereas array cannot provide this because it is fixed size.

Wrapper Class

The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*. The automatic conversion of primitive into an object is known as **autoboxing** and vice-versa **unboxing**.

Data Type	Wrapper Class
int	Integer
char	Character
float	Float
double	Double
long	Long
boolean	Boolean
	String (Already

Collection Framework



Generics

Generics are a powerful feature in Java that allows you to define classes, interfaces, and methods with type parameters. They enable type safety and reusability by ensuring that a collection or data structure works with specific types without requiring type casting.

1. **Type Safety**

```
List<String> list = new ArrayList<>();
list.add("Hello"); // Type-safe, only Strings allowed.
// list.add(123); // Compile-time error.
```

Generics

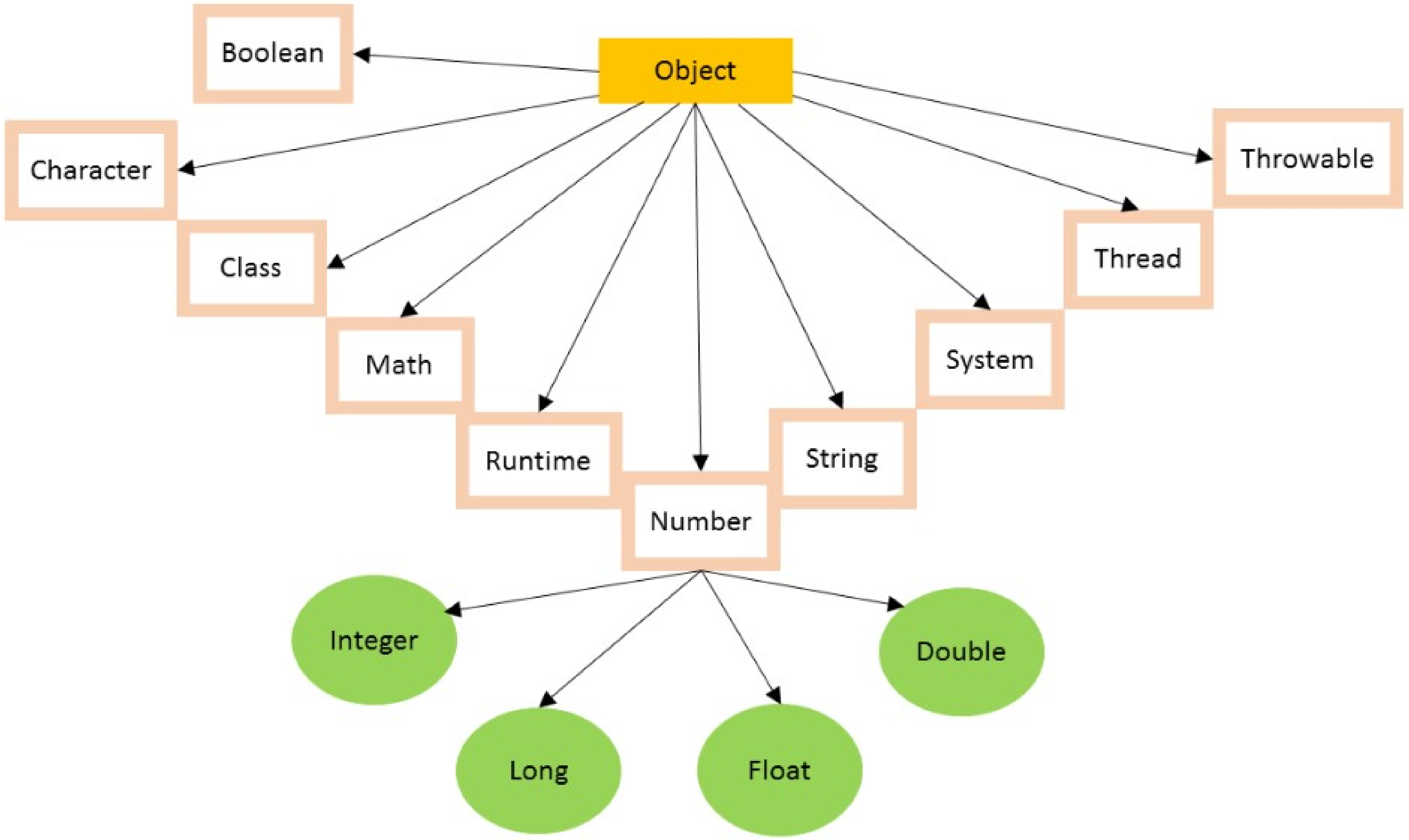
2. **Eliminates Type Casting:**

Without generics, type casting is required when retrieving objects from collections.

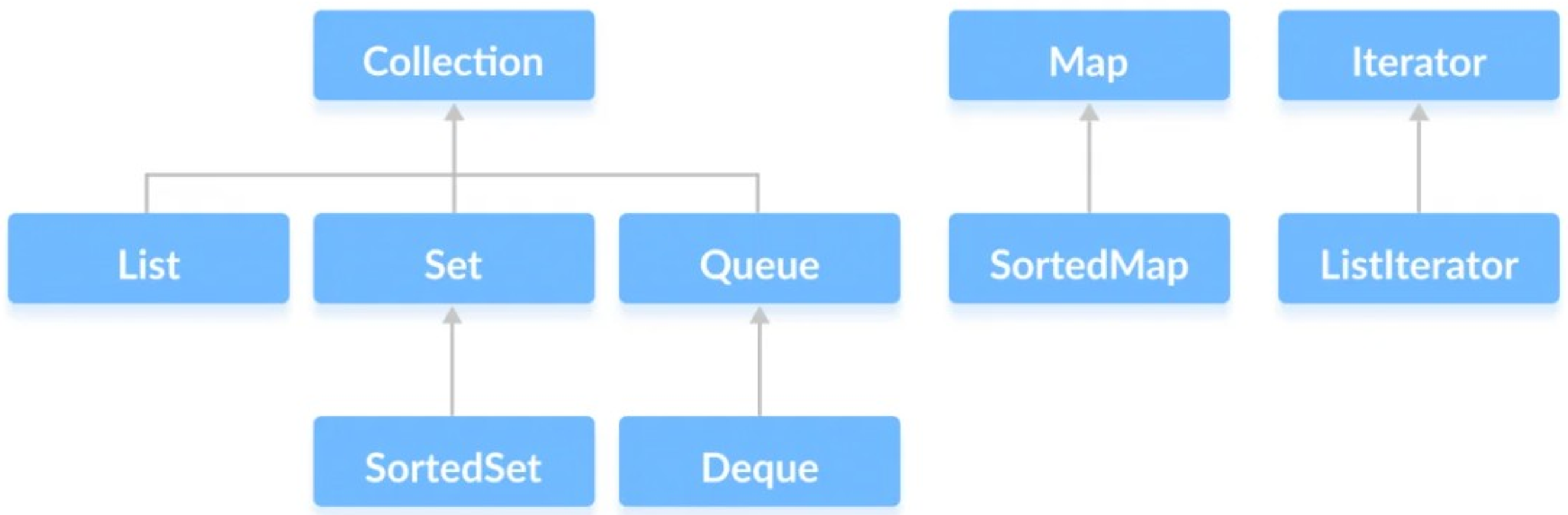
Ex:

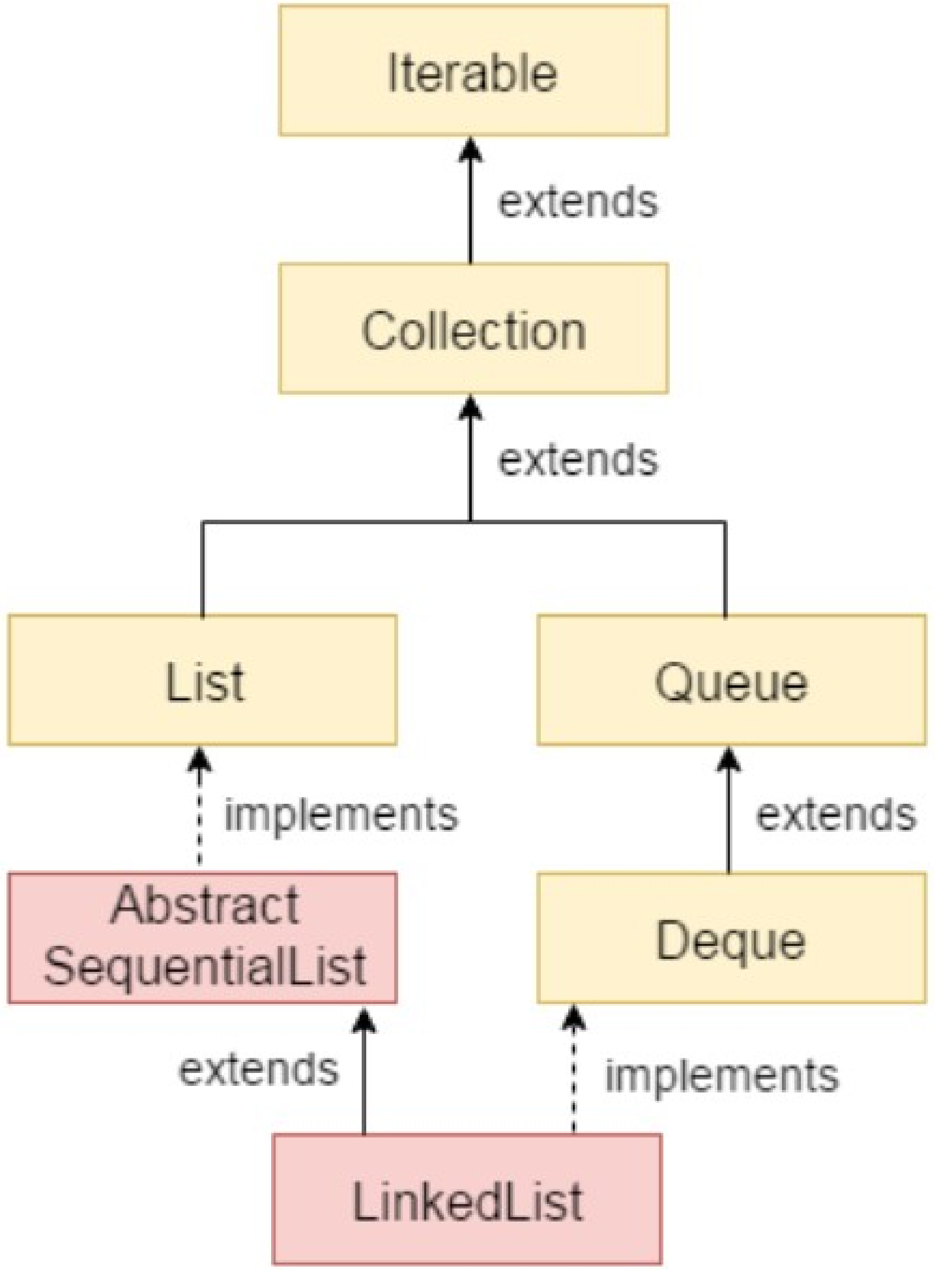
```
List list = new ArrayList();
list.add("Hello");
String str = (String) list.get(0); // Manual casting required.
```

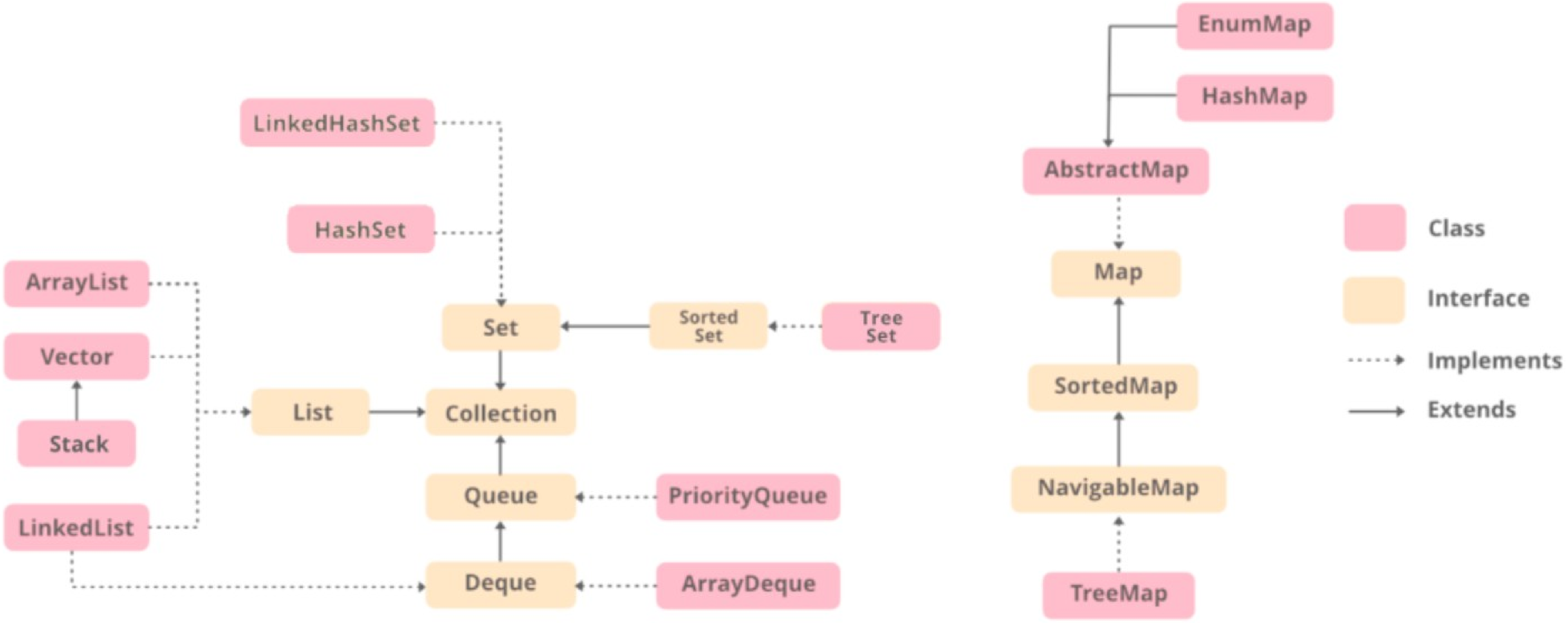
Object Class Hierarchy

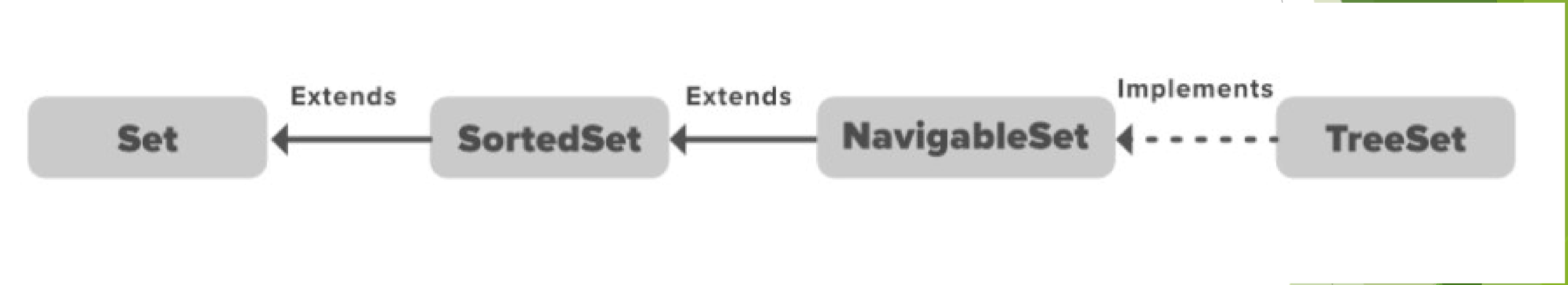


Java Collections Framework




Iterable
extends
Collection
extends
List
Queue
implements
extends
Abstract SequentialList
Deque
extends
implements
LinkedList


EnumMap
HashMap
LinkedHashSet
AbstractMap
HashSet
Class
ArrayList
Map
Set
Sorted Set
Tree Set
Interface
Vector
Implements
SortedMap
List
Collection
Extends
Stack
NavigableMap
Queue
PriorityQueue
LinkedList
Deque
ArrayDeque
TreeMap


Set
Extends
SortedSet
Extends
NavigableSet
Implements
TreeSet

List vs Set vs Map

List	Set	Map
List maintains insertion order	Set does not maintain insertion order	May is Key Value pair. Insertion order depends upon which map we use.
List allows the duplicate	Set does not allow duplicate	
Elements can be inserted at specified index	Elements cannot be inserted at specified Index. Because Set does not maintain index.	

Vector Vs Stack

A Vector in Java is a part of the Java Collections Framework and is a dynamic array that can grow or shrink in size as needed. It is similar to an ArrayList but with two key differences: it is synchronized, and it contains many legacy methods that are not part of the collections framework

A Stack is in collection framework which implements list interface and extends Vector class. It is based on the basic principle of last-in-first-out (LIFO). It has two basic operations called push and pop which is used to insert and remove the elements. In addition to that it also provide few more operations empty, search and peek opeations.

Vector Vs Stack

Key Differences

Feature	Vector	Stack
Data Structure	Dynamic array	LIFO (extends Vector)
Access	Random access by index	Access only to the top element
Order	Preserves insertion order	LIFO (Last In, First Out)
Usage	General-purpose, thread-safe	LIFO-based operations
Thread Safety	Synchronized	Synchronized (inherits from Vector)
Modern Alternative	ArrayList	Deque (such as ArrayDeque)

Stack Vs Vector

Key differences:

1. **Order of elements:** Stack (LIFO) vs. Vector (maintains insertion order).
2. **Access method:** Stack (push/pop) vs. Vector (get/set).
3. **Synchronization:** Both are synchronized, but Stack is more efficient.
4. **Resizeability:** Vector is resizable, while Stack has fixed size or overflow handling.

When to use:

5. **Stack:**
   - Evaluating postfix expressions.
   - Implementing recursive algorithms.
   - Parsing.
   - Backtracking

2. **Vector:**
   - Dynamic memory allocation.
   - Random-access requirements.
   - Array-like operations.

Deque

The **Deque** is related to the double-ended queue that supports adding or removing elements from either end of the data structure

Therefore, a deque can be used as a stack or a queue. We know that the stack supports the Last-In-First-Out (LIFO) operation, and the operation First-In-First-Out is supported by a queue. As a deque supports both, either of the mentioned operations can be performed on it.

Autoboxing and Auto Unboxing

Autoboxing is the automatic conversion that the Java compiler makes between **the primitive types and their corresponding object wrapper classes.** For example, converting an int to an Integer, a double to a Double, and so on. If the **conversion goes the other way, this is called unboxing.**

Collections

Iterator:

'Iterator' is an interface which belongs to collection framework. It allows us to traverse the collection, access the data element and remove the data elements of the collection.

java.util package has **public interface Iterator** and contains three methods:

1. **boolean hasNext()**: It returns true if Iterator has more element to iterate.
2. **Object next()**: It returns the next element in the collection until the hasNext()method return true. This method throws 'NoSuchElementException' if there is no next element.
3. **void remove()**: It removes the current element in the collection. This method throws 'IllegalStateException' if this function is called before next( ) is invoked.

List vs Set vs Map

List

- It is list of elements
- It is ordered collection of elements
- It allows duplicate values
- New items can be inserted anywhere in between the list

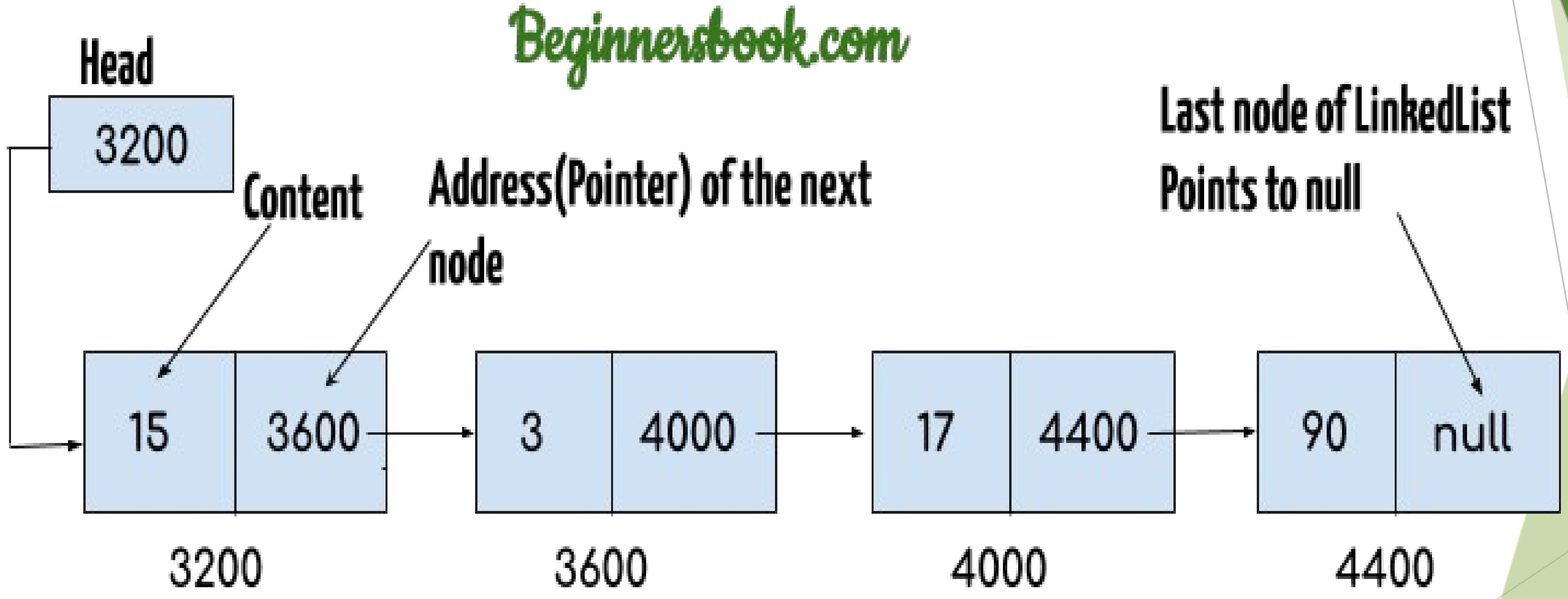
Set

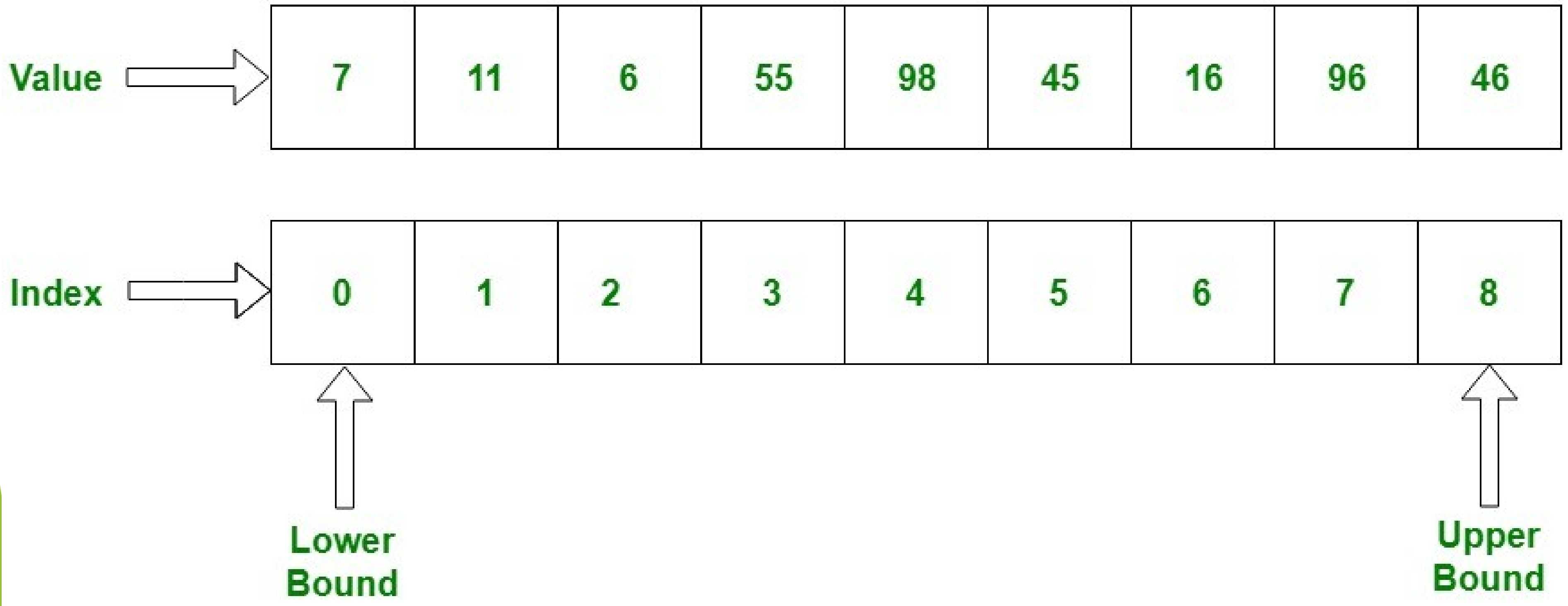
- It is set of elements
- It is not ordered collection of elements
- It does not allows duplicate values
- New items can not be inserted anywhere in between the set

Map

- It is Key-Value pair of elements
- It is not ordered collection of elements
- If duplicate key is given, it will be overridden.
- HashMap allows one null key and multiple null values. But HashTable does not allow null key and null value.

Linked List


Beginnersbook.com
Head
3200
Content
Address(Pointer) of the next node
Last node of LinkedList Points to null
15
3600
3
4000
17
4400
90
null
3200
3600
4000
4400



Array Length = 9

Read and Write Text File

1. Ways to read data from the file

```
BufferedReader br = new BufferedReader(new FileReader(file));
FileReader fr = new FileReader("C:\\Users\\pankaj\\Desktop\\test.txt");
Scanner sc = new Scanner(file);
InputStreamRead isr = new InputStreamReader();
```

2. Ways to write data to the file

```
BufferedOutputStream bos = new BufferedOutputStream(new
FileWriter(file));
FileOutputStream outputStream = new
FileOutputStream("c:/temp/samplefile4.txt");
DataOutputStream dataOutStream = new DataOutputStream(new
BufferedOutputStream(outputStream));
```

Read and Write Excel File

Apache POI – Read Write Excel File

What is Apace POI?

Apache POI (Poor Obfuscation Implementation) is a Java API for reading and writing Microsoft Documents in both formats **.xls** and **.xlsx.**

- **HSSF (Horrible SpreadSheet Format) Implementation:** It denotes an API that is working with Excel 2003 or earlier versions.
- **XSSF (XML SpreadSheet Format) Implementation:** It denotes an API that is working with Excel 2007 or later versions.

JDBC

There are 5 steps to connect any java application with the database use JDBC

1. Register the Driver class
2. Create connection
3. Create statement
4. Execute queries
5. Close connection

JDBC

```
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
con.close();
```

BLOB and CLOB

BLOB

Binary Large Object. It stores the binary data up to 2GB of size. This helps to store the image file, voice data.

CLOB

Character Large Object. It stores the character up to 2GB of size.

BLOB and CLOB

BLOB	CLOB
The full form of Blob is Binary Large Object.	The full form of Clob is Character Large Object.
This is used to store large binary data.	This is used to store large textual data.
This stores values in the form of binary streams.	This stores values in the form of character streams
Using this you can stores files like videos, images, gifs, and audio files.	Using this you can store files like text files, PDF documents, word documents etc.